Toward a TLM to RTL refinement: a formal approach

Jean-François Le Tallec[1,2] Julien DeAntoni[1]
[1]Université de Nice Sophia Antipolis, INRIA Sophia Antipolis Méditerranée,
06902 Sophia Antipolis, France
[2]Scaleo Chip, 1681 route des Dolines, Sophia Antipolis, 06560 Valbonne, France
{jfletall,julien.deantoni}@sophia.inria.fr

Abstract

Due to increasing complexity of SoC and shortening life time cycle of product, time to market becomes a major challenge in SoC design. To overcome this problem, an abstract representation of the platform under development can be used by software developers at the early stage of the development. This abstracted platform is then refined until its complete specification. For now, it remains difficult to implement and simulate a system that mixes TLM and RTL code. In this paper, we proposed a first step to bridge this gap by using CCSL *as a mean to synchronize the system interfaces independently of their abstraction level. We describe a potential way to logically specify transaction at different levels of abstraction.*

1. Introduction

The increasing complexity of embedded systems and the reduction of the product life cycle make time-to-market to rule SoC (System On Chip) development. Traditionally, SoC design flow goes through sequential steps including: platform design, platform implementation, software design, and software implementation. Consequently, the earlier a platform is available, the earlier software engineers can design and develop software code and the earlier the final product is available to the market. To improve this design flow, SoC companies start using concurrent conception approaches. More specifically, SoC design uses a so called *virtual platform*, which simulates the functionalities of the actual platform. Some years ago, virtual platforms simulated the low level behavior of the real platform. However the simulation time increased with the complexity of the platform thus a need for abstraction appeared. Consequently, virtual platform shifted from a low abstraction level representation of hardware (like classically done with common HDL (Hardware Description Language)) to an higher abstraction level of representation. An abstracted virtual platform focuses on message passing and hides the unimportant information about communication. This kind of abstraction is often referred to as TLM (Transaction Level Modeling) due to the SystemC initiative [6].

SystemC offers the ability to handle TL (Transaction Level) or RTL (Register Transfer Level) so that it becomes a good candidate to develop virtual platforms. However, it is difficult to develop a system at transaction level independently of a specific kind of communication.

In this paper, we propose a way to formally specify the interconnection between IPs (Intellectual Properties, *i.e.,* communicating black boxes) at different abstraction levels. This specification is first very abstracted and can then be refined to add platform specific characteristics. The interconnection specification relies on the notion of *interface* and communication *protocol* between these interfaces. Both are expressed in a formal language named CCSL (Clock Constraint Specification Language) [1]. CCSL specifications can then be simulated and used to check the integrity of the IPs interconnection independently of their abstraction level.

After a description of the related works, we provide a short introduction to CCSL. Then, we present our approach, named communication centric approach, and illustrate it through a simple example.

2. Related works

In order to ease SoC design, people shift from traditional low abstraction level HDL like Verilog or VHDL to a classical and tooled language named SystemC [6]. SystemC is a open source C/C++ library, which aims to describe both hardware and software components. Concept of time, event, sensitivity, signal, port, module and process have been defined to enable hardware description as general HDLs. SystemC takes benefits from existing C++ compilers and comes with its own simulator. An IP written in traditional HDL can be translated into its SystemC version even if an automated translation is not trivial [7]. In this case, the resulting SystemC code is still described at RTL level and no abstraction is realized.

In order to abstract SystemC IPs, an add-on named TLM has been developed. It specifies communication mechanisms at a high abstraction level called *transaction*. By using TLM, in one hand the system under development can be simulated faster due to the hiding of non meaningful communication protocol messages, on the other hand,

a system can be first specified at a high abstraction level and validated by simulation.

However, the notion of transaction was not clearly stated thus different philosophy appeared. While it assumes that it represents a highly abstract communication, implementations like the one provided by ARM [2] reflects a specific bus protocol. In this case, a refinement of a TLM specification imposes the use of the AMBA bus to be consistent and no other bus can be used. In the same manner, ST-Microelectronic proposed another view of the transaction named TAC (Transaction Accurate Protocol) [8] but its implementation remains bus specific as highlighted in GreenSoCs' review [3]. So, there is a clear lack on the definition of what a transaction is and what necessary information a transaction definition must carry to enable a real refinement, possibly leading to totally different implementation of the transaction at RTL level.

In order to allow formal verification of such systems, some work has been done around the formalization of SystemC [4]. Based on these works, a first drawback is the difficulty to formally define SystemC/TLM with a single paradigm. For instance, Maraninchi et al. [4] tried to give formal execution to SystemC/TLM in a synchronous and then in an asynchronous formalism. Because neither the first nor the second formalism fit to specify SystemC, they used an hybrid representation [9] with micMac automaton. However, this formal representation of SystemC quickly leads to state explosion, making the model too large for automatic verification. Niemann et. al also proposed an approach based on FSM [5] that integrates the SystemC scheduler.While very accurate, this representation only considers untimed transation level.

In the remainder of this paper we formalized our approach focused on transaction dependencies, aiming to allow refinement toward low description level, keeping activation order and potentially enhancing different level interconnections to interact with each other.

3. CCSL

CCSL (Clock Constraint Specification Language) is a formal specification language [1] initially introduced in the OMG UML profile for *Modeling and Analysis of Real-Time and Embedded* systems (MARTE). It defines a rich-but-well-defined variety of time (logical/physical, dense/discrete...) encapsulated in the notion of *clocks*. A clock refers to a set of *instants*, which is usually infinite and totally ordered. For each of its instant, a clock is said to *tick*. A discrete time clock can be attached to an event, and in this case, each *tick* of the clock represents an occurrence of the event. The clock ticks are not necessarily directly bounded to "physical" time, only the ordering matters, hence the given name of *logical clocks*.

CCSL relies on three fundamental binary relations on instants: *precedence* ($\preccurlyeq$), *coincidence* ($\equiv$), and *exclusion* ($\#$). The *strict precedence* ($\prec$) is a derived instant relation: $\prec \triangleq \preccurlyeq \setminus \equiv$. Clock relations are built on these elementary relations. Only the subset of CCSL used in this paper is briefly defined but the reader can refer to [1] for a complete and formal description.

coincidence, denoted $\boxed{=}$, is a strong synchronous relation that imposes a pairwise coincidence between the instants of each clock.

exclusion, denoted $\boxed{\#}$, specifies that not any instant of one clock can coincide with an instant of the other clock.

strict precedence, denoted $\boxed{\prec}$, is an asynchronous constraint that imposes the k^{th} instant of the left clock always precedes the k^{th} instant of the right clock.

strict alternation, denoted $\boxed{\sim}$ is a derived clock relation imposing a double precedence: $a \boxed{\sim} b$ iff $\forall k \in \mathbb{N}^{\star}, a[k] \prec b[k] \prec a[k+1]$.

CCSL also defines *clock expressions*. A clock expression defines a new clock from existing ones. A few of them are used in this paper:

union, denoted $+$, creates a new clock which ticks whenever an operand clock ticks. To ease reading, let

$$\bigcup_{i \leq M} Sig_i = (Sig_1 + Sig_2 + ... + Sig_M)$$

minus, denoted $-$ creates a new clock which ticks whenever the left clock ticks while the right clock does not.

We used the subset of clock relations and expressions presented above to specify various kinds of protocols and properties associated with IPs, thereby abstracting behavior.

4. Communication centric approach

The proposed approach is centered on the IP communications. Consequently, we defined a system as a set of interconnected IPs. IPs are seen as components, which possess interfaces through which they communicate with their environment. The approach aimed to validate a specific assembly of IPs at different abstraction levels. We first formalized the notion of transaction on an interface. Then, based on this definition, we provided a method to specify and to refine the communication protocol between the interfaces.

4.1. Interface

In our model each IP owns interfaces to communicate with the others. We differentiate two kinds of interfaces, the *master* and the *slave* interfaces. Master interfaces can only communicate with a slave interface and master interfaces are responsible for the initiation of the communication. This differentiation is motivated by the need to identify the orientation of a communication, and consequently, of a transaction. A transaction is defined by three

oriented logical signals whose activities are specified by CCSL clocks: *Req*, *Ack* and *Done*. From a master point of view, Req is an output while Ack and Done are inputs. From a slave point of view Req is an input while Ack and Done are outputs (signal orientation are mirrored between slave and master interfaces).

Req is a request for communication initiated by the master. Ack indicates the beginning of the communication. Done indicates the end of the communication. There are intuitive minimal relations between these signals. First of all, each request is sent before its acknowledgement. And then, an acknowledgement is sent before the end of the communication. In order to formalize these relations, we used CCSL so that, for both master and slave interfaces, a transaction is defined by three clocks and two specific relations as follow:

$$Req \boxed{\prec} Ack \qquad Ack \boxed{\prec} Done$$

This formal definition provides a very high abstracted view of a transaction on an interface. The next section details how the interconnection of two interfaces is formalized to realize an abstract communication.

4.2. Protocol

A protocol is inherently a set of constraints over signals exchanged during a specific communication. In our case, these constraints reflect the communication between a master and a slave interface. At the higher level of abstraction, a communication is considered to behave like a basic wire binding. Consequently, during the communication between a master and a slave interface, the following intuitive rules appear:

$$Req_{master} \boxed{=} Req_{slave}$$
$$Ack_{master} \boxed{=} Ack_{slave}$$
$$Done_{master} \boxed{=} Done_{slave}$$

With the formal definition of master and slave interfaces and the definition of the interconnection protocol in CCSL, it is possible to simulate the communication behavior of the system (see Figure 1). If there is no viable solution with respect to the specification, CCSL detects a deadlock and the simulation stops. Consequently, a simulation ensures the consistency of interconnection specification with respect to input scenarii. As highlighted by

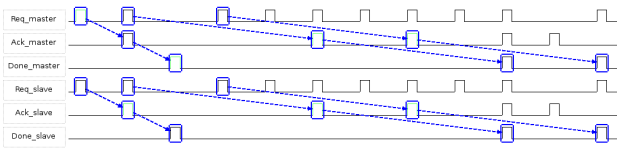

Figure 1. Simulation with unbounded queue

the Figure 1, at this level of abstraction we did not restrict the number of "pending" transactions. Consequently, a master can make an infinite number of request independently of the fact that the previous ones are finished or not. It can be seen as a communication with unbounded queues. To come closer to physical protocol implementations and, for instance, consider bounded queues, a refinement must be done.

4.3. Refining the communication

The refinement of the communication is a restriction of its possible behavior. Consequently, it is materialized by the addition of constraint over the protocol. We first illustrate this refinement to represent a communication with bounded queues.

The constraint to add must limit the number of emit signals depending of the number of finished communication and the queues size. Consequently, for a queue of size N, it constraints the number i of Req to be lower than the number j of Ack + N. Mathematically speaking, $\forall i, Ack[i] \prec Req[i+N]$. To easily illustrate this, we take a queue size of 1. consequently we add the following relations:

$$Req_{master}[i] \prec Ack_{master}[i] \prec Req_{master}[i+1]$$

The previous relations on instants are specified on clocks in CCSL by an alternation relation. Consequently, the associated relation is: $Req_{master} \boxed{\sim} Ack_{master}$. We also bind, for the slaves, the number of pending Ack by adding the same constraint than previously exposed, leading to the following relation: $Ack_{slave} \boxed{\sim} Done_{slave}$. With these relations, the communication queue is now bounded to one as highlighted by the simulation of the system in Figure 2. Consequently, the master can make one new request (Req) as soon as the previous one is started (Ack) and another transaction is started when previous one is finished (Done).

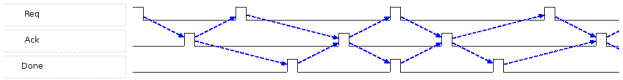

Figure 2. Simulation with bounded queue

$$\bigcup_{i\leq M} Req_{Slavei} \boxed{=} \bigcup_{j\leq N} Req_{Masterj}$$
$$\bigcup_{i\leq M} Ack_{Slavei} \boxed{=} \bigcup_{j\leq N} Ack_{Masterj}$$
$$\bigcup_{i\leq M} Done_{Slavei} \boxed{=} \bigcup_{j\leq N} Done_{Masterj}$$
$$\forall i \leq N, j \leq N, i \neq j, Req_{Masteri} \boxed{\#} Req_{Masterj}$$
$$\forall i \leq M, j \leq M, i \neq j, Req_{Slavei} \boxed{\#} Req_{Slavej}$$
$$\bigcup_{i\leq M} Req_{Masteri} \boxed{\sim} \bigcup_{j\leq M} Ack_{Masterj}$$
$$\bigcup_{i\leq M} Ack_{Slavei} \boxed{\sim} \bigcup_{j\leq M} Done_{Slavej}$$

Another refinement step can now consider the use of a shared medium between N masters and M slaves. This medium allows only one master to start a transaction and only one slave to be accessed at a time. Seven constraints are defined to represent a communication through this shared medium.The first three constraints ensure the propagation of all transactions from N masters to M slaves and generalize the previously presented P2P protocol for N masters and M slaves. The fourth and fifth constraints avoid concurrent accesses of the medium by different masters and slaves. The sixth and the seventh constraints generalize global interface relations. The constraints are then the following:

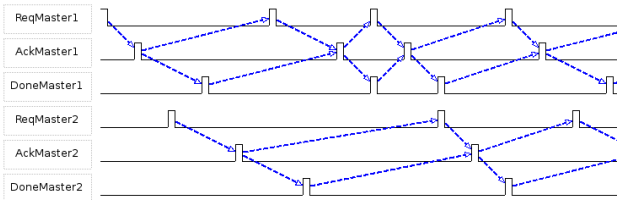

Figure 3. Simulation of two-master/two-slave system

Potential evolution of the system can be see on Figure 3. We can see that the last relation ensure that no overlapping between Ack and Done occurs (a medium is busy from an Ack and until a Done). Slave are not represented here but behave as masters do.

To represent more complex protocols such as the static priority protocol, we introduced intermediate signals for each master interface representing conflict-free requests in one case and delayed requests (due to simultaneous request with greater priority) in the other. They are respectively denoted *CFReq* and *DelayedReq*. From a slave point of view, the requests are now denoted *RealReq* and correspond to the union of conflict-free requests and delayed requests. Let the priority of a master be defined by $P(master)$. Then the static priority protocol for a two masters and one slave system can be described with the previous constraints increased with priority handling. Priority handling is specified by the following relations:

$$\forall i \leq N,\ CFReq_{Master i} \boxed{=} Req_{Master i} - \bigcup_{j,P(j)>P(i)} RealReq_{Master j}$$

$$\forall i \leq N, j \leq N, (Req_{Master i} - CFReq_{Master i}) \boxed{\sim} DelayedReq_{Master i}$$

$$\forall i \leq N\ RealReq_{Master i} \boxed{=} CFReq_{Master i} + DelayedReq_{Master i}$$

$$\forall i \leq N, RealReq_{Master i} \boxed{\sim} Ack_{Master i}$$

$$\forall i, j \leq N, i \neq j, DelayedReq_{Master i} \boxed{\#} RealReq_{Master j}$$

The first relation defines the conflict-free requests. The second relation specifies that less prior masters have their requests delayed when a conflict occurs. The third defines what the real *Req* from a slave point of view is. The fourth relation adapts relation previously stated on the new signal RealReq. Finally, the fifth relation forbids delayed requests to be in conflicts with real ones. The resulting simulation of such system where master 1 has a higher priority than master 2 is presented Figure 4 where the first request of the master 1 is in conflict with the second request of master 2. Due to its priority, master 1 is served before master 2 whose request is delayed. We then generalize these relations for multiple masters and multiple slaves to allow us to define behavior of hierarchical systems. They can be express by the conjunction of all previous relations.All relations mentioned previously refer to logical constraints. Using same kinds of relation, all these logical clocks can be mapped to match physical clocks. Then temporal activation of IPs described at different level is enhanced and timing performance of a system can be deduced.

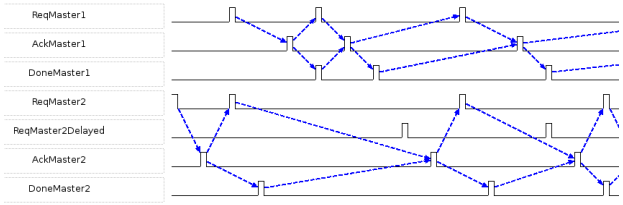

Figure 4. Simulation of static priority system

5. Future works and conclusion

We presented a formal way to specify communications between IPs at different level of abstraction. It starts with a very abstract specification of a transaction. In opposite to other identified approaches, the TLM representation is independent of any platform implementation details. Then, without any modification on the IP interfaces, the communication can be refined to add platform knowledge step by step. At each step, formal validation ensures consistency of the IP interconnections with regards to the specified protocol. Two future works directly extend such an approach. First a generalization could allow the specification of hierarchical systems. Second, a code analyzer could allow an abstract representation of IP behavior from a communication point of view.

References

[1] C. André. Syntax and Semantics of the Clock Constraint Specification Language (CCSL). Research Report RR-6925, INRIA, 2009.

[2] ARM: AMBA-PV extensions to OSCI TLM 2.0 documentation. http://infocenter.arm.com.

[3] Greensocs review of STMicroelectronics TAC protocol. http://www.greensocs.com/fr/projects/TACPackage/greensocs_review_st_tac.pdf.

[4] F. Maraninchi, M. Moy, J. Cornet, L. Maillet Contoz, C. Helmstetter, and C. Traulsen. SystemC/TLM Semantics for Heterogeneous System-on-Chip Validation. Montréal Canada, 06 2008.

[5] B. Niemann and C. Haubelt. Formalizing tlm with communicating state machines. In *FDL*, pages 285–293, 2006.

[6] Open SystemC Initiative: SystemC 2.2 / TLM 2.0 documentation. http://www.systemc.org.

[7] J. Ruf, D. Hoffmann, J. Gerlach, T. Kropf, W. Rosenstiehl, and W. Mueller. The simulation semantics of SystemC. In *DATE '01: Proceedings of the conference on Design, automation and test in Europe*, pages 64–70, Piscataway, NJ, USA, 2001. IEEE Press.

[8] St microelectronics: Transaction accurate communication, 2005. http://www.greensocs.com/TACPackage.

[9] C. Traulsen, J. Cornet, M. Moy, and F. Maraninchi. A SystemC/TLM semantics in Promela and its possible applications. In *14th Workshop on Model Checking Software SPIN*, July 2007.